PWM Design Brief

Our PWM_Tmer module is a multi-channel, programmable PWM and timer generator with Wishbone bus support. It supports 4 independent PWM outputs with individually configurable duty cycles and periods, controlled either via internal registers or an external duty cycle input.

Key Features:

- **4 PWM Channels** (o_pwm[3:0]) with:
  - Separate Period and Duty Cycle registers
  - Optional external DC input (i_DC)
- **Dual Clock Source:**
  - Select between system clock (i_clk) and external clock (i_extclk) via control bit
- **Clock Divider:**
  - Adjustable clock rate using Divisor
  - Handling Invalid Divisor Values
- **Dual Operation Modes:**
  - PWM Mode: Generates pulse-width-modulated signals
  - Timer Mode: Acts like a timer with interruption including the ability to determine which channel caused interruption and one-shot/continuous run
- **Wishbone Interface:**
  - Supports register reads/writes via i_wb_* signals
- **Safe DC Handling:**
  - Clips Duty Cycle to not exceed Period (safe_DC)

Tricky Challenges:

1- **Challenge:** Managing multiple PWM outputs with distinct Period/DC values while sharing the clock and control logic.
Solution: We used a generate block to iterate over the 4 channels (genvar ch), creating per-channel logic with shared clocking. Each channel uses: Its own main counter[ch], DC [ch], Period[ch] Shared Ctrl settings and clock divider This avoids code duplication and ensures consistent timing across all channels while maintaining configurability.

2- **Challenge:** In PWM and timer systems, invalid settings — such as a zero divisor, or a duty cycle greater than the period — can lead to unstable behavior, glitches, or even total failure of signal generation.
Solution: We implemented built-in safety mechanisms to detect and correct these issues at runtime Duty cycles are automatically clamped to the value of the period to prevent incorrect PWM output

3- **Challenge:** While building the testbench, we realized that manually writing Wishbone transactions and control sequences for each scenario (e.g., setting period, duty cycle, switching modes, testing interrupts or down-clocking) would lead to *repetitive and lengthy code*, making the testbench harder to manage and extend.

Solution: We addressed this by creating reusable tasks and functions. These abstractions allowed us to replace large code blocks with simple, modular calls, improving readability and scalability. This structure made it easier to add or modify test cases without cluttering the codebase or duplicating logic.

Test Sequence:

The PWM timer testbench validates a PWM/Timer module with Wishbone bus interface. It runs 7 comprehensive tests covering PWM generation, timer interruptions, clock division, and error handling.

Test 1: Wishbone Operations: Validates the Wishbone interface by writing and reading back random values to all registers, checking read data equal to the written one.

Test 2: PWM 50% Duty Cycle: Tests standard PWM generation with a period of 100 cycles and 50% duty cycle. Enables full PWM output and verifies the signal alternate between high and low states for equal durations.

Test 3: PWM 25% Duty Cycle: Demonstrates real-time duty cycle modification by changing from 50% to 25% while PWM is running. Shows the system can dynamically adjust PWM parameters without stopping operation.

Test 4: Timer Interrupt Generation and Clearing: Set the Timer mode, the interrupt flag is correctly set to 1 upon the counter reaching its Period. Then Confirms the interrupt clearing mechanism by writing a 0 to the interrupt flag bit and checking the flag is successfully reset. This happens in one-shot and continuous timer modes

Test 5: Clock Divisor /2: In Timer mode testing the timer at slower rate with a divisor 2. It ensures the timer's interruption pulse is generated after a duration that is precisely two times longer than the Period (Interrupt occurs after $20 \times 2 = 40$ clock cycles)

Test 6: Clock Divisor /10: In Timer mode testing the timer at slower rate with a divisor 10. It ensures the timer's interruption pulse is generated after a duration that is precisely ten times longer than the Period (Interrupt occurs after $20 \times 10 = 200$ clock cycles)

Test 7: Edge Cases: Comprehensive boundary condition testing with three sub-tests:

- Tests duty cycle greater than period (expects PWM to stay constantly high)
- Tests divisor of zero (expects system to operate as if divisor is 1 for protection)
- Tests manual counter reset functionality during operation

GitHub Project Repo Link : https://github.com/MohGendy/PWM-Timer